The HPS3 service: reduction of cost and transfer time for storing data on clouds

Jorge Veiga, Guillermo L. Taboada, Xoán C. Pardo, and Juan Touriño
Computer Architecture Group
University of A Coruña
Campus de Elviña s/n, 15071
A Coruña, Spain
Email: {jorge.veiga, taboada, pardo, juan}@udc.es

***Abstract*—In the past several years, organizations have been changing their storage methods as the volume of data they managed has increased. The cloud computing paradigm offers new ways of storing data based on scalability and on good conditions of reliability and accessibility. This paper proposes the design of *HPS3*, a service that uses compression and concurrency techniques in order to reduce storage costs and data processing times in public storage providers. Different strategies for compressing and uploading data depending on differential characteristics of the datasets are also explained. The evaluation of *HPS3* shows, in comparison with the default use of the cloud storage provider (Amazon S3), an average reduction of 61.6% in data transfer volume, 85.5% in upload time and 73.2% in incurred costs. Compared to Dropbox, *HPS3* shows an average improvement of 27.4% in data transfer volume and 93.6% in upload time.**

***Keywords*—Cloud Computing, Cloud Storage, Data Compression, Concurrency, Amazon S3, Dropbox**

I. Introduction

The amount of data stored in the cloud is growing fast. Besides the use of cloud storage from particular users, there are more and more companies and research centers that decide to locate their data with cloud services due to their accessibility and price features.

The use of cloud computing for storage shows, however, some problems the user has to deal with, such as security, network latency, waste of bandwidth or underutilization of computing resources. There are several public storage providers where particular users store their data: Amazon S3, Dropbox, Google Drive, Microsoft SkyDrive, etc. However, there are some unmet demands from organizations with a higher level of requirements in terms of transfer time and incurred costs.

The cost of storing files is usually determined by the occupied space in the cloud provider, the number of requests and the data transfer volume. The occupied space and the data transfer volume can be reduced by using compression. The number of requests can be reduced by aggregating files in groups, compressing each group to one archive file and uploading it using a single request. The considered groups can be, typically, the own directories in which the dataset may be divided.

The aim of this paper is to analyse how data can be stored in cloud storage providers optimizing costs and reducing the execution time of data processing operations. As a result, a cloud-based storage service called *HPS3* (which stands for High Performance S3) is proposed, with a flexible design based on concurrency and compression.

The structure of this paper is as follows: Section II presents the related work. Section III explains the proposed solution for a high performance storage service in the cloud. Section IV describes the experimental configuration used in the evaluation and presents comparative results with well-known cloud storage services using representative datasets. Finally, Section V summarizes our concluding remarks.

II. Related Work

Storage in the cloud is an active research topic that is gaining increasing attention. Besides services that only use public storage providers, some works study the way to use cloud storage for enhancing existing storage systems. FriendBox [1] uses a hybrid Friend-to-Friend (F2F) storage system, combining resources of trusted friends with cloud storage to improve QoS of pure F2F systems. The use of cloud storage allows the service to increase performance and data availability, mitigating the drawbacks of pure decentralization. The paper also

includes an analysis of the costs of FriendBox in terms of use of the cloud.

Costs of storing the data generated by workflow-based scientific applications are analysed in studies such as [2], which proposes an algorithm to store large application datasets using linear Data Dependency Graphs (DDGs). The strategy developed in the paper automatically selects several datasets from a larger group to be stored in the cloud. The rest of datasets will be regenerated from the stored ones when needed. The applicability of this approach is limited to the kind of applications to which it is oriented (where the DDG is known).

Other works focus on providing optimized solutions for encrypting and storing data in the cloud, like Cryptonite [3]. This secure data repository is the result of the study of several optimizations in the encryption of data to be uploaded to Windows Azure. Many ideas in the paper can be applied to compression when uploading large datasets to storage providers, allowing the service to reduce storage costs. This reduction can also be achieved by studying which public storage provider is best to use. In order to do this, an integrated storage service called iCostale [4] uses data compression, cloud pricing schemes and history of access patterns. The framework optimizes the storage costs taking into account specific characteristics of each dataset and even storing its data using different providers. Deployed as a cloud service, iCostale does not reside with the client, which involves an additional overhead in the communications with the storage provider.

Cumulus [5] is a cloud-based backup system. It aggregates small files for remote storage, segments the files and implements incremental changes in order to maintain storage efficiency. The paper shows how aggregation can reduce the number of requests to the cloud storage provider, which reduces costs and increases the compression ratios. However, the system does not take into account differential characteristics of the uploaded datasets and, despite using concurrency, the paper does not show the way it is applied.

III. The HPS3 Storage Service

The main goal of the service to be implemented is to achieve the best possible efficiency and to reduce the pricing of the storage provider used. The service is intended to be used for enterprise backup or storage of data to be processed in the cloud provider, and is targeted for both enterprise and research environments. This section covers the analysis of the service scenario, the different compression strategies that have been considered and the resulting design.

A. Service Scenario

The main objective of *HPS3* is to offer a fully functional cloud storage service that optimizes cost savings and transfer times. Both terms can be improved by using data compression when performing data operations, which are divided in upload and download operations. Upload operations consist in compressing the data and uploading it to the provider. Download operations consist in downloading the compressed data and decompressing it. The achieved compression ratio depends on several factors from which the following ones are taken into account: (1) Compression algorithm, (2) Data type, and (3) Dataset structure.

First, the chosen compression algorithm will show a certain compression ratio and efficiency (time taken to compress the data). Although a higher compression ratio will involve higher cost savings in the target cloud provider, the compression operation may be more time consuming. Therefore, the selection of the algorithm must take into account both terms. Second, the compression ratio of the selected algorithm may vary depending on the data type that is being compressed. Third, files can be compressed in several ways, one by one or by groups. The particular structure of the dataset may affect the optimal way of compressing the data. Compressing the files in groups can reduce the number of requests but it would make random accesses more difficult (the whole group must be downloaded to access a single file). Groups in which the files will be compressed can have different sizes. If the size is too big the latency of random accesses will be higher, and if the size is too small the number of requests will increase. In this paper, different compression strategies have been considered and will be explained in the next subsection.

Besides using data compression, the time taken by upload and download operations can be reduced by using concurrency, enabling the service to manage more than one file at a time. For example, to upload a directory that is divided in groups of files, each of these groups can be allocated to a processing instance, and the set of processing instances can be executed concurrently.

B. Compression Strategies

In order to take into account the distinct characteristics of datasets uploaded to the storage provider, different

strategies for compressing and uploading are proposed. It is assumed that each dataset will be composed by a file hierarchy that is going to be uploaded to the storage provider. The data will be downloaded and restored later or will be processed using cloud services, no matter the encoding used. The considered strategies are the following ones:

- *Plain*: The elements in the first level of the root directory are compressed and sent to the cloud provider. Redundancy among files is exploited, and full branches of the hierarchy are uploaded with a single request. This means fewer operations (compression plus upload), but larger transfers per request. A drawback that must be considered is the difficulties in random accesses: to download a single file it is necessary to download the full branch in which it is contained. Therefore, this strategy can be appropriate for compressing datasets which constitute a logical element by themselves.
- *Recursive*: The hierarchy is accessed until its last level, in which every single file is compressed and uploaded using one operation. Redundancy is only considered within each file, so the compression ratio can be lower in comparison with the plain strategy. It will also involve more operations, but allowing a higher level of concurrency. The random access does not add overhead, since every file can be recovered uniquely.
- *Mix*: It consists in the application of a recursive strategy until a user-selected level, in which the directories are fully compressed and uploaded to the storage provider. The mix strategy is presented as a tradeoff between the plain strategy (which provides reduction of operations and higher compression ratios) and the recursive strategy (which provides random access to the files and higher concurrency).

C. Design

An optimized storage service, *HPS3*, has been implemented using Amazon S3 as cloud storage provider. Amazon S3 is part of the AWS (Amazon Web Services) group of cloud services and allows to store files in the cloud in good conditions for durability and reliability. The files stored in Amazon S3 are represented as objects, each one contained in a bucket and identified by a unique key. Each bucket is located in a certain AWS Region and objects in the bucket never leave the region unless the user moves them.

HPS3 runs as a client connected to Amazon S3, allowing the user to display and manage files. These files can be contained in the local filesystem or in the cloud provider, and the user can modify their location by uploading and downloading them. These operations are handled by a pipeline that can be broken down in different subsystems as follows.

a) Connection: The Connection subsystem manages the connection between the client and the public storage provider. Once the connection is started, this subsystem makes the requests to manage the files and to retrieve the information about the stored data. This subsystem maintains dependencies with the API given by the public cloud provider, which allows to make the requests using high-level operations. Low-level issues like authentication, error management and retries are handled by the underlying libraries of the Amazon API.

b) Compression: The Compression subsystem contains the hierarchy of available compression algorithms: gzip, bzip2, tar.gz, tar.bz2, zip, rar, etc. Each algorithm can have multiple implementations in order to select the most suitable one for the environment in which the service may be running. This subsystem also provides the mechanisms to select a compression algorithm to be applied to a file. This selection is based on user-defined configuration options that link a certain data type to its most suitable compression algorithm.

c) Concurrency: The Concurrency subsystem allows *HPS3* to reduce the time taken to perform upload and download operations. To do this, this subsystem divides the source directory in different groups that are managed individually. These groups are, typically, the proper subdirectories in which the source directory may be divided. The different ways to divide a directory are defined by the compression strategies explained in Sec. III-B.

Once the directory is divided, each group of files is allocated to a processing instance. Each processing instance uses the Compression and Connection subsystems to compress and transfer, respectively, its group of files. The entire operation, consisting of a set of processing instances, is executed by a pool of threads. The number of instances that are processed concurrently is determined by the size of this pool. In order to make use of the full capabilities of the client machine, the implementation of

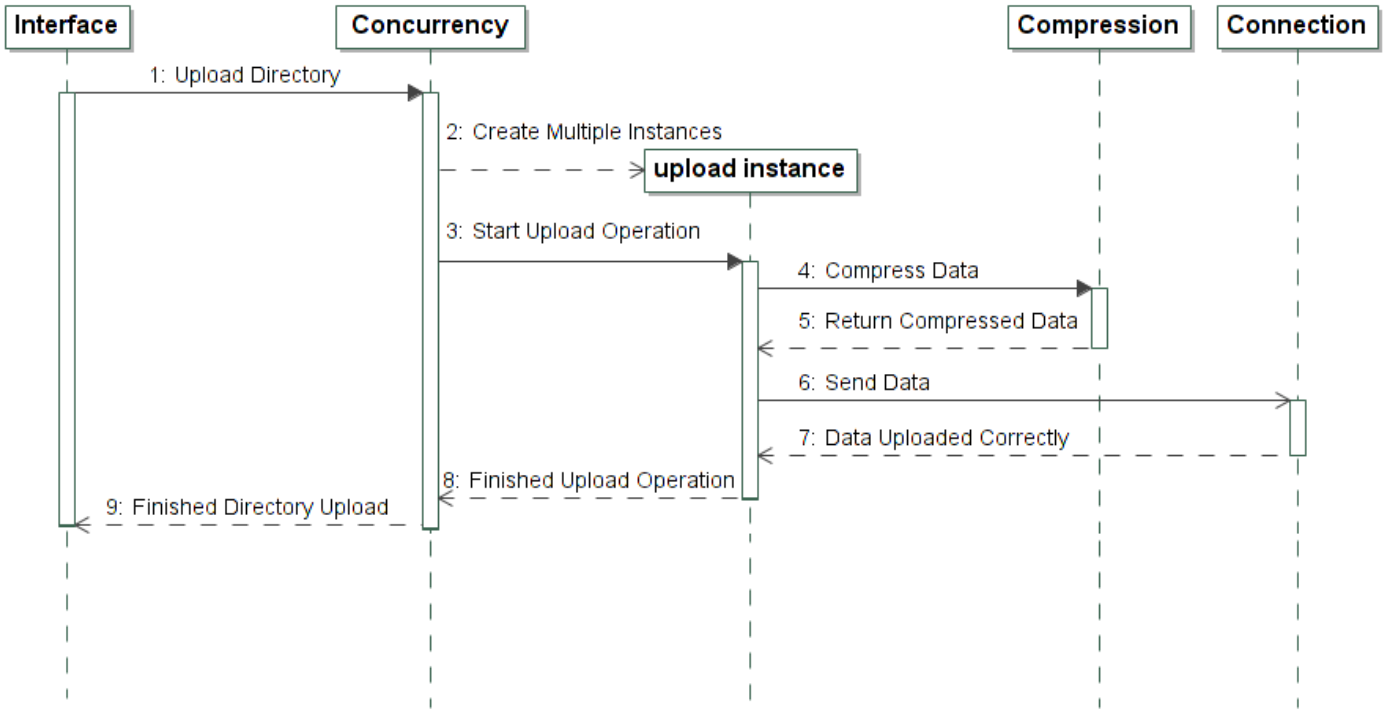


Fig. 1: Interaction among *HPS3* subsystems

HPS3 uses a pool of as many threads as the number of available cores.

d) Interface: The Interface subsystem allows the interaction of the user with the service, showing a view of local files and objects stored in the cloud provider. The user can upload and download files, as well as retrieve information about the files stored in the cloud.

At the time of uploading and downloading file hierarchies, the subsystems will communicate with each other to perform the action. Every subsystem contains the implementation of each function needed in the process. As an example, Figure 1 shows the sequence diagram of subsystem interactions for uploading a directory. The interaction shown in the diagram begins when the user selects to upload a directory in the Interface subsystem. The Concurrency subsystem reads the file hierarchy and creates an upload instance for every group of files in which the hierarchy is divided. This division is determined by the user-selected strategy. Each instance compresses the data using the Compression subsystem and uploads the compressed data using the Connection subsystem. Downloading a directory would work in a similar manner, but using download instances that would download the data and decompress it.

IV. Experimental Results

In order to evaluate the performance of the proposed solution, several upload experiments have been conducted. This section shows and discusses these results in terms of data transfer volume and data upload time.

A. Experimental Configuration

Four representative datasets (see Table I) have been selected to be uploaded using *HPS3*:

- *BOE*: a set of PDF documents from the Spanish Official State Bulletin (BOE), a daily release that publishes Spanish laws and public acts [6].
- *MEFF*: a set of economic data files from the Financial Futures Spanish Market (MEFF) [7].
- *DICOM*: a set of medical images in DICOM format [8].
- *OpenJDK*: the source code of Java version 6 [9].

The buckets used to upload the data are located in the AWS standard region, which is the US East (Northern Virginia) region. The client was run on an 8-core Intel® i7 CPU 930 at 2.80GHz with 6 GB RAM.

TABLE I: Characteristics of the datasets

Dataset	Total size	Number of files	File format	Hierarchy depth	Average file size
BOE	477.1 MB	852	PDF	4	556 KB
MEFF	643.2 MB	223	plain text	2	2.88 MB
DICOM	856.5 MB	80	DICOM	5	10.71 MB
OpenJDK	261.7 MB	25408	Java code	15	10.3 KB

The compression algorithm used in the experiments was tar.gz as it showed the best compression-time ratio in previous tests. Different strategies have been compared in terms of data transfer volume and upload time, using the same computer and network connection. The final results were obtained as the average values from a set of 10 experiments.

B. Performance Evaluation

In order to show the benefits of compression and concurrency in the upload operation, the sequential upload of the uncompressed data (i.e. the default scenario) has been compared with the different compression strategies described in Sec. III-B.

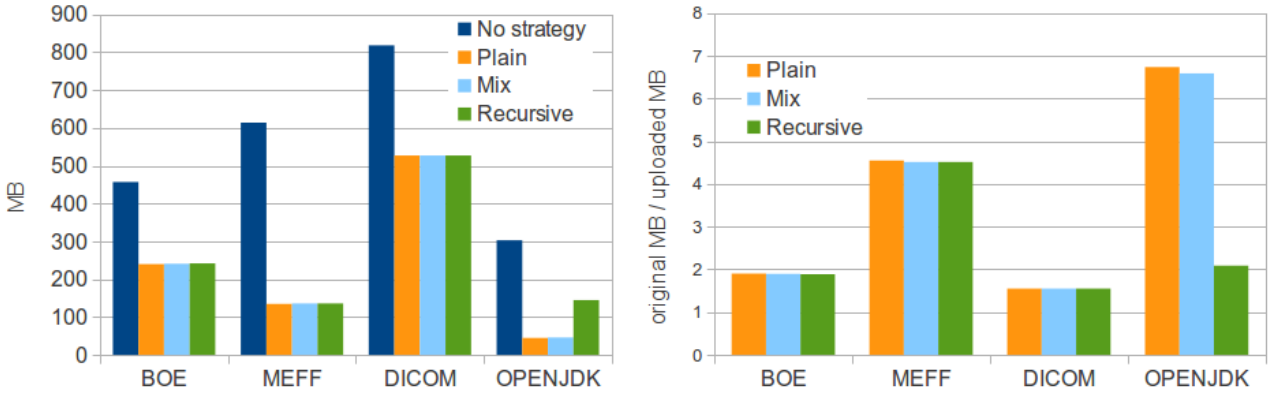
Figure 2a shows the size of the uploaded data and Fig. 2b the compression ratio for each dataset. The selection of the compression strategy is the only factor that affects the data transfer volume (concurrency has no influence on it). As can be observed in the results, there are not great differences between strategies for the BOE, MEFF and DICOM datasets, which suggests that redundancy among files is not being exploited in these datasets. However, the OpenJDK dataset has significant differences between strategies, due to the high redundancy that this dataset shows among its files. Using the plain or the mix strategies, which exploit this redundancy, the compression ratios are much higher than those of the recursive strategy, which only uses redundancy within each single file. Moreover, the DICOM dataset has the lowest compression ratio because it uses the JPEG format (which already includes compression) to encode the images. These results show that *HPS3* reduces the data transfer volume in a range from 35.6% (DICOM) to 85.1% (OpenJDK with the plain strategy).

Figure 3a shows the time taken for the same uploads, comparing each compression strategy with the sequential uploading without compression. These results include both compression and transfer times (except for the sequential case, which has no compression time) for each whole dataset. In this case, both the compression strategy and the use of concurrency (set to 8 threads, one per core) affect the upload times. Figure 3b shows the same results in terms of speedup relative to the sequential upload with no compression (i.e. the default scenario) and thus it is calculated as the quotient of the default upload time and the time using each compression strategy.

Some conclusions can be drawn from the results. DICOM is the dataset that gets the lowest speedup, which is 2.3 in the recursive case. This is caused by the fact that DICOM is the dataset with the largest files and the lowest compression ratio (as shown in Fig. 2b). Large files are more affected by interruptions, since an interrupted upload involves the data to be transmitted from scratch. This is due to the behaviour of the used public storage service, Amazon S3, which does not consider partial uploading of the objects.

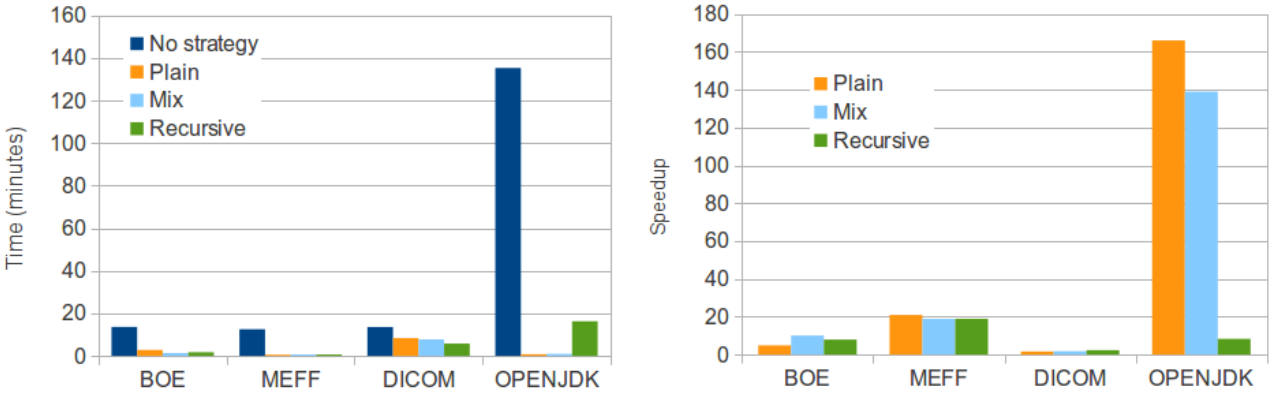
The BOE and MEFF datasets show good speedups. In the case of BOE, mix is the best strategy with a speedup of 10. MEFF does not show great differences among strategies, with a speedup near 20 (obtained with the plain strategy). OpenJDK has the most successful results. On the one hand, the speedup is 166 and 138 using the plain and mix strategies, respectively. On the other hand, the speedup is 8 with the recursive strategy. The performance using the plain strategy can be explained by the properties of the dataset, which consists of a large number of small files. Also, the directory tree has a large depth. The compression of each hierarchy branch to a single file causes a high reduction in the number of requests to the cloud provider, and increases the compression ratio (as shown in Fig. 2b). The use of the recursive strategy shows a larger upload time compared with the other strategies because compression and uploading have to be executed separately for every file of the dataset. In conclusion, according to the upload times of Fig. 3a, *HPS3* achieves a reduction from 57.4% (DICOM with the recursive strategy) to 99.4% (OpenJDK with plain).

In order to summarize the benefits of using *HPS3*,



(a) MB transferred to Amazon S3 (b) Compression ratio

Fig. 2: *HPS3* performance in terms of data transfer volume



(a) Upload time (b) Speedup

Fig. 3: *HPS3* performance in terms of upload time

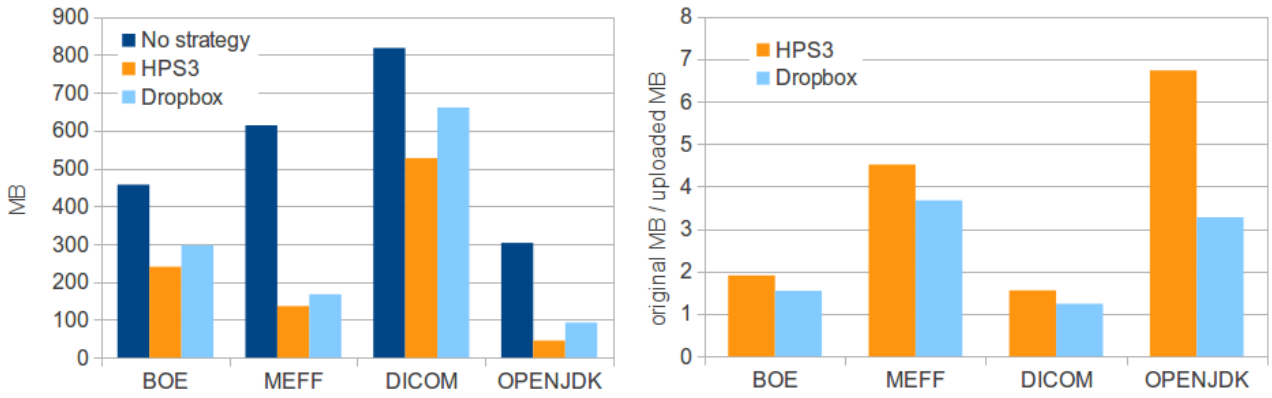
TABLE II: Summary of results for *HPS3*

	BOE	MEFF	DICOM	OPENJDK	Average
Speedup	10.1	20.9	2.3	166	48.7
Compression ratio	1.9	4.5	1.6	6.7	3.7
Cost savings ($)	50.3%	78.3%	35.4%	97.2%	73.2%

Table II shows the main features calculated for each dataset: speedup, compression ratio and cost savings, as well as the average result for each feature. The selected compression strategy has been the one that obtained the best upload time for each dataset, i.e. mix for BOE, plain for MEFF and OpenJDK, and recursive for DICOM. Cost savings have been calculated using Amazon S3 price rates [10] as for January 2014, considering the costs of each request and the volume of stored data when uploading the datasets.

C. Comparison with Dropbox

Dropbox is one of the most popular public storage services nowadays. It is directed to particular users, showing good performance in uploading modified files [11], due to the use of delta compression. This compression allows to upload only the modified parts of the files, reducing significantly the communication cost for dynamic datasets. Dropbox uses Amazon S3 for storing the data, behaving as an intermediary between the user and



(a) MB transferred (b) Compression ratio

Fig. 4: Data transfer volume using Dropbox and *HPS3*

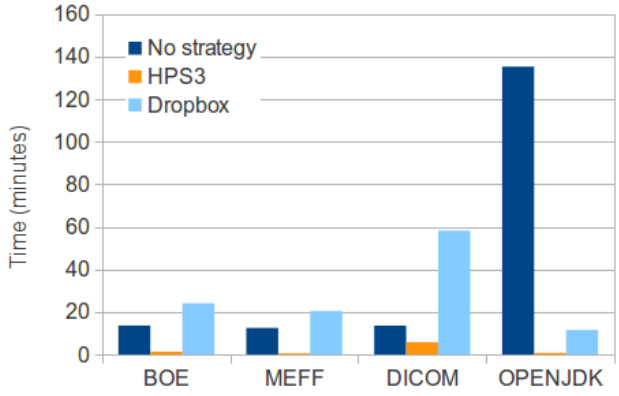


Fig. 5: Upload time using Dropbox and *HPS3*

the storage provider. This section shows a performance comparison of uploading our test datasets using Dropbox and *HPS3*. For *HPS3*, the compression strategy that obtained the best results was used for each dataset (see last paragraph of Sec. IV-B). As the datasets of the experiments are static, the compared behaviour of both systems may vary when managing dynamic datasets.

Figure 4a shows the volume of data uploaded to the cloud using each service and Fig. 4b the compression ratios achieved. As can be observed, these ratios are higher using *HPS3*, specially for the OpenJDK dataset, where Dropbox uses a recursive strategy that does not take advantage of the redundancy among files. The data transfer volume using *HPS3* is reduced with respect to Dropbox in a range from 18.7% (MEFF) to 51.4% (OpenJDK).

A comparison of the upload times is presented in Fig. 5 (note that the results for "No strategy" and "HPS3" are the same as in Fig. 3a). *HPS3* clearly outperforms Dropbox, reducing the upload time in a range from 90.1% (DICOM) to 97.1% (MEFF). These results can be explained by the following features of Dropbox:

- Lower compression ratio due to the delta compression, which introduces an overhead that penalizes the first upload.
- Recursive strategy to compress directories, which involves the execution of an upload operation (compression and transfer of the data) for every file in the dataset. This may involve a significant overhead when uploading datasets with a large number of files.
- Background client: unlike *HPS3*, the Dropbox client runs sequentially in the background, giving priority to the applications the user may be using.

- Multiple server connection: Dropbox executes an intermediate process for uploading the initial versions and deltas of the files. This process includes connections with several servers for uploading data and metadata, increasing the upload time. This is not the case of *HPS3*, which is executed completely on the client side.
- Transmission of large files: as said in Sec. IV-B, the transmission of large files is affected by interruptions. This issue gains significance in the case of Dropbox, since the upload involves connections with several servers.

V. Conclusions

This paper shows the benefits of using compression and concurrency when storing data on a cloud provider, having selected Amazon S3 as a case study. A design for a new service (*HPS3*) has been proposed, achieving a maximum reduction of 85.1% in data transfer volume, 99.4% in upload time and savings in cost up to 97.2% (61.6%, 85.5% and 73.2% reduction on average, respectively). Also, a comparison was performed between *HPS3* and Dropbox, one of the most popular cloud storage services. This comparison has shown a reduction of the data transfer volume when using *HPS3* up to 51.4% (27.4% on average), and a maximum reduction of 97.1% in upload time (93.6% on average).

Results have shown that the use of compression and concurrency can improve significantly the operations when storing large amounts of data on the cloud. The compression of files can reduce the number of requests to the cloud provider, and decrease the storage cost and the time to perform upload and download operations. In order to obtain the best results when uploading each particular dataset, several compression strategies have been studied, implemented and tested. The compression algorithm used in these strategies can vary depending on the data type of each dataset, which increases the flexibility of the service. The operation time is also reduced by the use of concurrency, by performing each upload or download operation using a separate thread.

The implemented service *HPS3* is fully operational and can be used for uploading and downloading data to Amazon S3 by any AWS account, providing its AWS credentials. The service can be configured by the user to determine which strategy to apply in every upload operation, and which compression algorithm to use for each type of file. Functionality can be extended in several ways. Firstly, encryption of the data can be provided by replacing the plain client by an encryption client, allowing the users to use their own keys to encrypt data before sending it through the network. Secondly, the interfaces that operate on the data could be adapted to other public storage providers by using open storage standards like CDMI (Cloud Management Data Interface) [12].

Acknowledgments

This work was supported by an Amazon Web Services Research Grant, by the Ministry of Economy and Competitiviness of Spain (project TIN2013-42148-P), and by the Galician Government (Xunta de Galicia) under the Consolidation Program of Competitive Reference Groups (Ref. GRC2013/055), cofunded by FEDER funds of the EU.

References

[1] R. G. Tinedo, M. S. Artigas, A. Moreno-Martínez, and P. G. López, "FriendBox: A hybrid F2F personal storage application," in *Proceedings of the IEEE 5th International Conference on Cloud Computing*, 2012, pp. 131–138.

[2] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A local-optimisation based strategy for cost-effective datasets storage of scientific applications in the cloud," in *Proceedings of the IEEE 4th International Conference on Cloud Computing*, 2011, pp. 179–186.

[3] A. G. Kumbhare, Y. Simmhan, and V. K. Prasanna, "Cryptonite: a secure and performant data repository on public clouds," in *Proceedings of the IEEE 5th International Conference on Cloud Computing*, 2012, pp. 510–517.

[4] S. Agarwala, D. Jadav, and L. A. D. Bathen, "iCostale: adaptive cost optimization for storage clouds," in *Proceedings of the IEEE 4th International Conference on Cloud Computing*, 2011, pp. 436–443.

[5] M. Vrable, S. Savage, and G. M. Voelker, "Cumulus: filesystem backup to the cloud," *ACM Transactions on Storage*, vol. **5**, no. 4, pp. 14:1–14:28, 2009.

[6] "Official State Bulletin," [Online]. Available:http://www.boe.es.

[7] "Financial Futures Spanish Market," [Online]. Available:http://www.meff.es/.

[8] "Digital Imaging and Communications in Medicine," [Online]. Available:http://dicom.nema.org/.

[9] "Java OpenJDK," [Online]. Available:http://openjdk.java.net/.

[10] "Amazon S3 Pricing," [Online]. Available:http://aws.amazon.com/es/s3/pricing/.

[11] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside Dropbox: understanding personal cloud storage services," in *Proceedings of the 12th ACM SIGCOMM Conference on Internet Measurement*, 2012, pp. 481–494.

[12] Storage Networking Industry Association (SNIA), "Cloud Data Management Interface," [Online]. Available:http://www.snia.org/cdmi.